Explaining Compressed Suffix Arrays Further.

From: Grossi, R., & Vitter, J. S. (2005). Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. SIAM Journal on Computing, 35(2), 378–407. https://doi.org/10.1137/S0097539702402354

The Suffix Array

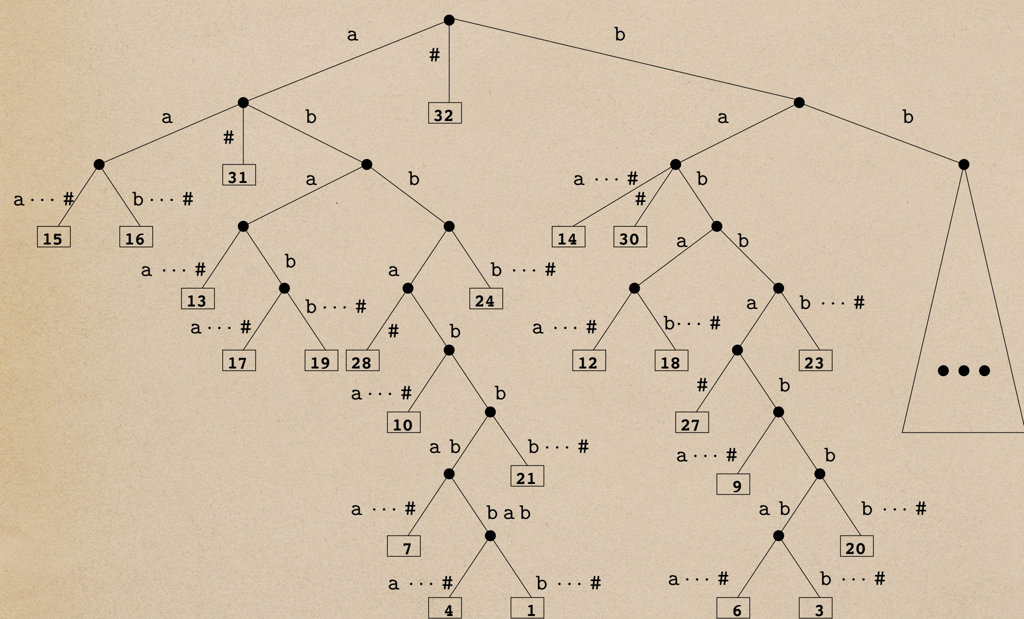

Figure 1: Suffix tree built on text T = abbabbabbabbabaaabababbabbbabba# of length $n = 32$, where the last character is an end-of-string symbol #. The rightmost subtree (the triangle representing the suffixes of the form bb · · · #) is not expanded in the figure. The edge label a · · · # or b · · · # on the edge leading to the leaf with value ℓ denotes the remaining characters of the suffix $T[\ell, n]$ that have not already been traversed. For example, the first suffix in lexicographic format is the suffix $T[15, n]$, namely, aaababababbabbbabba#, and the last edge represents the 16-symbol substring that follows the prefix aa.

1	15	aaababababbabbbabba#
2	16	aababababbabbbabba#
3	31	a#
4	13	abaaababababbabbbabba#
5	17	abababbabbbabba#
6	19	ababbabbbabba#
7	28	abba#
8	10	abbabaaabababbabbbabba#
9	7	abbabbabaaabababbabbbabba#
10	4	abbabbabbabaaabababbabbbabba#
11	1	abbabbabbabbabaaabababbabbbabba#
12	21	abbabbbabba#
13	24	abbbabba#
14	32	#
15	14	baaababababbabbbabba#
16	30	ba#
17	12	babaaababababbabbbabba#
18	18	bababbabbbabba#
19	27	babba#
20	9	babbabaaabababbabbbabba#
21	6	babbabbabaaabababbabbbabba#
22	3	babbabbabbabaaabababbabbbabba#
23	20	babbabbbabba#
24	23	babbbabba#
⋮	⋮	⋮
32	25	bbbabba#

Figure 2: Suffix array for the text T shown in Figure 1, where a < # < b. Note that the array values correspond to the leaf values in the suffix tree in Figure 1 traversed in in-order.

Suffix Array Decomposition

aaaa#	aaab#	aaba#	aabb#	abaa#	abab#	abba#	abbb#
12345	12354	14253	12543	34152	13524	41532	15432
baaa#	baab#	baba#	babb#	bbaa#	bbab#	bbba#	bbbb#
23451	23514	42531	25143	34521	35241	45321	54321

We use the intuitive correspondence between suffix arrays of length n and binary strings of length $n-1$. According to the correspondence, given a suffix array SA, we can infer its associated binary string T and vice versa. To see how, let x be the entry in SA corresponding to the last suffix `#` in lexicographic order. Then T must have the symbol `a` in each of the positions pointed to by $SA[1]$, $SA[2]$, …, $SA[x-1]$, and it must have the symbol `b` in each of the positions pointed to by $SA[x+1]$, $SA[x+2]$, …, $SA[n]$. For example, in the suffix array $\langle 45321 \rangle$ (the 15th of the 16 examples above), the suffix `#` corresponds to the second entry 5. The preceding entry is 4, and thus the string T has `a` in position 4. The subsequent entries are 3, 2, 1, and thus T must have `b`s in positions 3, 2, 1. The resulting string T, therefore, must be `bbba#`.

2.1 Decomposition scheme

Our decomposition scheme is by a simple recursion mechanism. Let SA be the suffix array for binary string T. In the base case, we denote SA by SA_0, and let $n_0 = n$ be the number of its entries. For simplicity in exposition, we assume that n is a power of 2.

In the inductive phase $k \geq 0$, we start with suffix array SA_k, which is available by induction. It has $n_k = n/2^k$ entries and stores a permutation of $\{1, 2, \ldots, n_k\}$. (Intuitively, this permutation is that resulting from sorting the suffixes of T whose suffix pointers are multiple of 2^k.) We run four main steps to transform SA_k into an equivalent but more succinct representation:

Step 1. Produce a bit vector B_k of n_k bits, such that $B_k[i] = \mathbf{1}$ if $SA_k[i]$ is even and $B_k[i] = \mathbf{0}$ if $SA_k[i]$ is odd.

Step 2. Map each $\mathbf{0}$ in B_k onto its companion $\mathbf{1}$. (We say that a certain $\mathbf{0}$ is the *companion* of a certain $\mathbf{1}$ if the odd entry in SA associated with the $\mathbf{0}$ is 1 less than the even entry in SA associated with the $\mathbf{1}$.) We can denote this correspondence by a partial function Ψ_k, where $\Psi_k(i) = j$ if and only if $SA_k[i]$ is odd and $SA_k[j] = SA_k[i] + 1$. When defined, $\Psi_k(i) = j$ implies that $B_k[i] = \mathbf{0}$ and $B_k[j] = \mathbf{1}$. It is convenient to make Ψ_k a total function by setting $\Psi_k(i) = i$ when $SA_k[i]$ is even (i.e., when $B_k[i] = \mathbf{1}$). In summary, for $1 \leq i \leq n_k$, we have

$$\Psi_k(i) = \begin{cases} j & \text{if } SA_k[i] \text{ is odd and } SA_k[j] = SA_k[i] + 1; \\ i & \text{otherwise.} \end{cases}$$

Step 3. Compute the number of $\mathbf{1}$s for each prefix of B_k. We use function $rank_k$ for this purpose; that is, $rank_k(j)$ counts how many $\mathbf{1}$s there are in the first j bits of B_k.

Step 4. Pack together the even values from SA_k and divide each of them by 2. The resulting values form a permutation of $\{1, 2, \ldots, n_{k+1}\}$, where $n_{k+1} = n_k/2 = n/2^{k+1}$. Store them into a new suffix array SA_{k+1} of n_{k+1} entries, and remove the old suffix array SA_k.

The following example illustrates the effect of a single application of Steps 1–4. Here, $\Psi_0(25) = 16$ as $SA_0[25] = 29$ and $SA_0[16] = 30$. The new suffix array SA_1 explicitly stores the suffix pointers (divided by 2) for the suffixes that start at even positions in the original text T. For example, $SA_1[3] = 5$ means that the third lexicographically smallest suffix that starts at an even position in T is the one starting at position $2 \times 5 = 10$, namely, `abbabaa...#`.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
T:	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0:	15	16	31	13	17	19	28	10	7	4	1	21	24	32	14	30	12	18	27	9	6	3	20	23	29	11	26	8	5	2	22	25
B_0:	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
$rank_0$:	0	1	1	1	1	1	2	3	3	4	4	4	5	6	7	8	9	10	10	10	11	11	12	12	12	12	13	14	14	15	16	16
Ψ_0:	2	2	14	15	18	23	7	8	28	10	30	31	13	14	15	16	17	18	7	8	21	10	23	13	16	17	27	28	21	30	31	27

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
SA_1:	8	14	5	2	12	16	7	15	6	9	3	10	13	4	1	11